Presented at IEEE International Conference on Neural Networks, 3/28 - 4/1/93, San Francisco

Genetic Algorithm based Input Selection for a Neural Network Function Approximator with Applications to SSME Health Monitoring[†]

Charles C. Peck and Atam P. Dhawan
Dept. of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, OH 45221

Claudia M. Meyer
Sverdrup Technology, Inc.
NASA Lewis Research Center Group
Brook Park, OH 44142

***Abstract*— A genetic algorithm is used to select the inputs to a neural network function approximator. In the application considered, modeling critical parameters of the Space Shuttle Main Engine, the functional relationships among measured parameters is unknown and complex. Furthermore, the number of possible input parameters is quite large. Many approaches have been proposed for input selection, but they are either not possible due to insufficient instrumentation, are subjective, or they do not consider the complex multivariate relationships between parameters. Due to the optimization and space searching capabilities of genetic algorithms, they were employed in this study to systematize the input selection process. The results suggest that the genetic algorithm can generate parameter lists of high quality without the explicit use of problem domain knowledge. Suggestions for improving the performance of the input selection process are also provided.**

I. INTRODUCTION

There is considerable interest within the space industry in improving the fault detection and isolation capabilities of rocket engine condition monitoring systems, both real-time and post-test. This requires developing accurate models of engine parameters based on other measured parameters. Developing accurate models is particularly difficult due to the highly complex, non-linear nature of rocket engines, the limited suite of measured parameters, and the large variability of behavior among engines of the same design.

It has been shown that neural networks with one hidden layer can uniformly approximate any continuous function [1, 2, 3]. Furthermore, neural networks are well-suited for problems in which the exact relationships between inputs and outputs are complex or unknown [4, 1]. These conclusions may be applied to dynamical systems if the system state is sufficiently represented in the inputs of the neural network. For these reasons, feedforward neural networks have been used to model critical parameters of the Space Shuttle Main Engine (SSME) during the start-up transient and they have been shown to be effective [4].

A task that is critical to the success of neural network modeling of complex, dynamical systems such as the SSME is the choice of input parameters. There are several constraints that complicate this task. First, while the instrumentation of the SSME is extensive, it is not complete. Therefore, it is unlikely that it will be possible to completely describe any subsystem input or output. This makes the use of characteristic equations particularly difficult [5]. Second, as was discussed above, it is necessary to provide enough state information to model the desired parameter. Finally, it is not practical to use a large number of inputs for the following reasons. First, large input sets make training the network considerably more difficult. In addition, the input set should be small to reduce the hardware and/or computational complexity and to reduce the processing delay. This last consideration is especially important if the system is to be used for real-time modeling.

The effect of the input size on the system performance is further exacerbated by the use of multiple past values (or time windows) of each input. A time window of each input parameter is typically used in order to provide time dependent information. The size of the window multiplies the number of inputs to the network. For example, if 10 parameters are chosen as network inputs and a time window of the past ten values is used for each parameter, then the effective number of inputs to the network is 100.

Many approaches for input selection have been proposed or used. These include characteristic equations, engine schematic analyses, correlations between candidate input parameters and the modeled parameter, and expert advice [5]. As suggested above, the use of some of these methods is limited due to the lack of sufficient instrumentation. Furthermore, some of these methods are subjective or they do not adequately measure the multivariate dependencies present in the system. For these reasons, a systematic approach for input selection under the existing

[†]This work was supported by a contract from the NASA Space Engineering Center for System Health Management Technology at the University of Cincinnati

constraints is desired.

The choice of inputs may be modeled as an optimization problem where the space of possible solutions is quite large. In fact, many hundreds of sensors are used for monitoring during test firings of the SSME. If only 100 sensors were used, then 2^{100} distinct input sets would exist. Since an exhaustive search through a space with this order of magnitude is clearly not possible, an alternative search method is required.

Genetic algorithms are well suited for searching in a large parameter space [6, 7]. Through the use of seeding (the process of providing an initial set of possible solutions), genetic algorithms search from a set of solutions or starting points, rather than a single starting point. Genetic algorithms are not derivative based, thus they can search spaces where methods such as conjugate descent fail. They work with both discrete and continuous parameters. They explore the parameter space and exploit the similarities between highly fit candidate solutions [6, 8]. Furthermore, through the use of elitism (a variant method in which the best solution of a generation is promoted unaltered to the next generation), a genetic algorithm can be guaranteed to perform at least as well the methods used to seed or initialize it. For these reasons, a genetic algorithm was used to select the inputs to a neural network that modeled an SSME parameter during the start-up transient.

This paper will first present the design issues and methodology applied to the selection of SSME input parameters. A presentation and discussion of results will follow. Finally, the conclusions and ideas for future work will be presented.

II. DESIGN ISSUES AND METHODOLOGY

There are three fundamental design requirements for applying genetic algorithms: encoding candidate solutions onto binary strings, decoding and evaluating a binary string, and developing a fitness function that will guide the genetic algorithm to produce the desired results.

For this application, encoding candidate solutions onto binary strings is trivial since a single bit is sufficient to indicate whether a particular parameter is to be included in the network input set. Accordingly, the string, or *chromosome,* has one bit for every candidate engine parameter. To reduce the size of the search space, redundant sensor measurements were eliminated and those parameters believed to be nearly independent of the modeled parameter were not included in the candidate parameter set. This reduced the size of the candidate parameter set to 49 parameters from the hundreds of parameters described above.

Decoding and evaluating a chromosome is also straightforward. First, the number of parameters encoded in the chromosome is determined. Then, a neural network with the proper number of inputs is created and the weights are randomly initialized. Each network is provided with ten hidden layer nodes. A training set is created using the parameters indicated in the chromosome. The input sets are expanded to include a time window of five past values for each parameter. Finally, the neural network is trained. To limit the computational requirements in training the neural networks, the QuickProp learning algorithm is used [9] and each network is trained for only 100 epochs, which was determined empirically to be sufficient to distinguish the performance of one network configuration from another. According to the analysis provided in [9], this should be comparable to 1000 epochs of training with standard backpropagation.

The fitness function must be able to guide the genetic algorithm to produce the smallest set of input parameters that are sufficient for a neural network function approximator to accurately predict a modeled parameter. Ideally, the resulting neural network function approximator should be able to learn and generalize the relationships between the input parameters and the output parameter such that the approximation is accurate even for test firings not encountered during training. Of these two network performance objectives, the determination of learning ability is the most straightforward and the only one considered in this paper. To provide the guidance described above, the fitness function must measure some properties of the candidate solution and compute a number that indicates the "fitness" of that solution. The properties considered in this paper include the performance of the generated neural network, the convergence properties of the training process, and the number of inputs. It should be noted prior to discussing fitness function implementation that, in this paper, the smaller the fitness function value, the better the evaluated solution is considered to be.

The performance of the neural network is determined by measuring the approximation error of the network on the training data. This provides a measure of how well a network is capable of learning the input/output relationships.

As mentioned above, the convergence properties of the training process are also considered in the fitness function. In this manner either early convergence or late convergence can be favored. This is important since, as described above, the neural networks are not fully trained during the genetic algorithm in order to reduce the computational requirements. To understand the relevance, consider two networks, one that has a low training error and early convergence, and another that has a higher training error and late convergence. It is not clear which network should be considered better. It may be argued that the first network is better since it converged to a stable condition rapidly. Conversely, it may be argued that the second network is better since it is still in a rapid learning phase

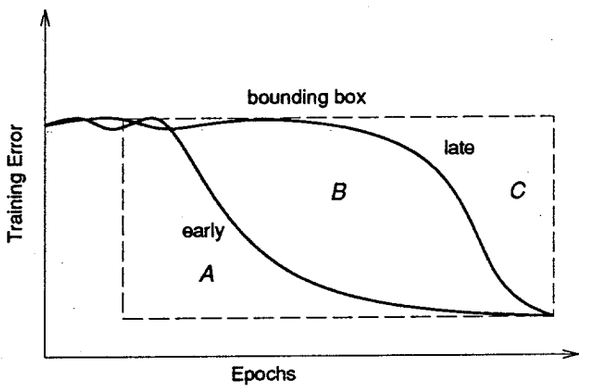

Figure 1: Early versus Late Training Error Convergence

and, with further training, its error performance may exceed the performance of the first network. Both of these cases were used in the overall system to evolve input parameter sets.

The method used to reflect the convergence properties exploits the observation that the training error of each neural network began on a high plateau and remained there, at least briefly, before falling rapidly, as shown in Figure 1. Since oscillations and unusual patterns in the training error were not observed, integration of the area bounded by the error curve and a bounding rectangle could be performed. To favor early convergence, the convergence term is computed by integrating this area of integration and normalizing it by the area of the bounding box. For example, if A, B, and C denote the normalized areas of their corresponding regions in Figure 1, the convergence term is A for the early training error curve and $A + B$ for the late training error curve. The convergence term favoring late convergence is simply the complement of the convergence term favoring early convergence. This corresponds to a convergence term of $B + C$ for the early training error curve and C for the late training error curve.

As described above, the fitness of a candidate solution is also related to the size of the input parameter set. Including the input set size constraint in the fitness function could be done simply by multiplying the training error by the number of parameters selected. This, however, results in a very strong constraint. The strength of the size constraint can be controlled by adding a constant to the number of parameters selected. A small offset created in this manner yields a strong size constraint, whereas a large offset yields a weak one. The fitness function may be further adjusted by squaring the size constraint term. This increases the strength of the constraint as the number of parameters increases.

The design of the size constraint term in the fitness function is further complicated by the size disparity between the chromosomes within the seeding population. The initial population of the genetic algorithm was seeded in two ways. First, four sets of input parameters were selected based on prior knowledge of SSME behavior. These seeding sets consist of approximately 10 parameters each. The remainder of the initial population was generated with the use of a random number generator, which selected each input parameter with a probability of 50%. Thus, the randomly selected seeding sets consist of approximately 25 parameters each. If all of the seeding sets were approximately the same size, a single offset could be chosen that would yield the desired input set size at the end of the evolution process. The size disparity, however, between the knowledge-based seeding sets and the randomly selected seeding sets results in either a strongly biased choice of input parameters or it results in input sets that are too large.

For the work presented in this paper, generation dependent offsets were used to avoid biasing the results while ensuring satisfaction of the size constraint. Initially, the offset was set very high to allow the candidate solutions to compete primarily on the basis of the training error. As the genetic algorithm proceeded, the size constraint was made progressively stronger. By the last generation the offset was small, yielding a strong bias for shorter lists. This change of offset with respect to the generation will be referred to as an offset progression. Two offset progressions were used: one yielding a generally weak size constraint, and another yielding a generally strong size constraint. The offset progression yielding the weaker size constraint ranged from 71 initially to 14 over 20 generations. The other ranged from 45 initially to 7 over 20 generations. The resulting fitness functions are shown in Equations 1 and 2, respectively:

$$f_{\text{weak}} = C\frac{(c+71-3G)^2}{(71-3G)^2} \times \text{Training Error}, \quad (1)$$

$$f_{\text{strong}} = C\frac{(c+45-2G)^2}{(45-2G)^2} \times \text{Training Error}, \quad (2)$$

where f is the fitness function value, C is the convergence term (which may be 1.0 if no convergence properties are considered), c is the number of parameters in the candidate input list, and G, which ranges from 0 to 19, is the generation number.

The process for selecting sets of input parameters proceeded in two stages. The first stage consisted of independently evolving three different populations of candidate solutions. One population was evolved using a fitness function without any training convergence bias. The other two were biased for early and late training convergence. Each of these populations was evolved with a weak size constraint to favor lower approximation error.

The ten most fit chromosomes from each of the three first stage populations were used along with 20 randomly

generated chromosomes to seed the second stage genetic algorithm. The primary purposes of the second stage were to merge the three diverse and independent populations, and to further reduce the size of the parameter lists. To meet this last objective a fitness function with a strong size constraint and no convergence bias was used.

The use of three independent first stage populations increases diversity, robustness, and resistance to domination by "Super Individuals." "Super Individuals" are suboptimal solutions that are significantly more fit than other solutions early in the evolution process. This allows them to dominate a subsequent population after a few generations. The concept of "Super Individuals" should not be confused with elitism. Elitism only guarantees that the best string will be promoted to the next generation unaltered, it does not imply dominance.

Since the fitness functions described above are *noisy*, they provide less guidance than deterministic fitness functions. They are noisy because the approximation errors of a particular network will vary from the errors of other implementations of the same network. Implementations of a particular network vary because they are each initialized with random weights. Although different implementations perform similarly after training, they perform differently due to their generally unique final weight sets.

When noisy fitness functions are present, the *generational replacement* technique is typically used [7] to mitigate the undesirable effects of inaccurate fitness evaluations. Generational replacement is implemented by completely replacing the population each generation. In addition to using generational replacement in this study, each chromosome was reevaluated whenever it occurred in a population. Combined, these techniques "temporally" average the effects of fitness function variations on chromosomal representation in future populations.

III. RESULTS AND DISCUSSION

The fundamental output of the genetic algorithm consists of candidate parameter lists to be used as inputs to a neural network for modeling a particular parameter during start-up. The parameter that was modeled is the SSME's High Pressure Oxidizer Turbine (HPOT) discharge temperature, which has a Parameter IDentification (PID) number of 233. The three parameter lists generated by training-based fitness functions with the best fitness values from the last generation of the second stage are presented in Table 1. These three lists, labeled GA-1–GA-3, have been shown to be physically reasonable in the context of the SSME [10]. An additional list, labeled REF in Table 1, is also presented for the purpose of comparison. This "reference" list has been modified from the one presented in [4] to exclude autoregressive information. The PID's in each of these lists are described in Table 2.

It should be noted that the knowledge-based seeding lists were outperformed early in the process by genetic algorithm generated parameter lists. This may be explained by the larger number of input parameters included in the randomly selected seeding sets and the lack of a strong size constraint in the early generations. The larger sets of parameters had smaller training errors and they were not penalized for their size. While the behavior and results of the genetic algorithm were certainly affected by the knowledge-based seeding sets, the guidance provided by these sets did not appear to be strong.

To evaluate the performance of the parameter lists produced by the genetic algorithm and the reference list, feedforward neural networks were trained for 20,000 cycles using the standard backpropagation learning algorithm. Each network had one hidden layer with 10 hidden units and used a time window of five past values. The resulting networks were then used to approximate PID 233 using measured parameters from 12 actual SSME test firings. Four of the test firings were used for training the networks and eight were used to validate the resulting models. The results, as represented by the mean squared error (MSE), the normalized MSE, and the maximum percent error, are shown in Tables 3, 4, 5, and 6. A summary of these results is presented in Tables 7 and 8. The results are divided into two groups: one presenting the aggregate performance of the networks on the training data (Table 7), and the other presenting the aggregate performance of the networks on the validation data (Table 8). The first group measures the learning capabilities of the networks and the second group measures the generalization capabilities.

In considering the performance of the different networks, it should be recognized that even though the networks are fully trained, their final performance is still dependent on their weight initializations. Thus, the performance of another network implementation may differ. It is clear from the results that the parameter list GA-1 has the worst error performance of the four lists. This is compensated by the fact that this is the shortest parameter list. Even though the error performance of this parameter list is the worst, it is still close to the performance of the other lists, including the reference list.

The parameter lists GA-2 and GA-3 outperformed the reference list on the training data and performed only slightly worse than the reference list on the validation data. Due to the large standard deviations of validation data error, the differences in the error means between these two particular network implementations cannot be considered statistically significant. Thus, it can concluded that the parameter lists GA-2 and GA-3 perform approximately as well as the reference list.

IV. CONCLUSIONS AND FUTURE WORK

The results indicate that the error performance of the genetic algorithm generated parameter lists is roughly the

Table 1: Parameter Lists

Parameter List	*Number of PIDs*	*Parameters*
GA-1	6	21 58 209 734 951 1050
GA-2	7	21 58 209 327 734 951 1058
GA-3	8	21 52 58 209 327 734 951 1050
REF	9	40 42 59 231 480 1205 1212 O/Cs OPBs

Table 2: Parameter Descriptions

PID	*Description*
21	Main Combustion Chamber Oxidizer Injection Temperature
40	Oxidizer Preburner Oxidizer Valve Actuator Position
42	Fuel Preburner Oxidizer Valve Actuator Position
52	High Pressure Fuel Pump Discharge Pressure
58	Fuel Preburner Chamber Pressure
59	Preburner Boost Pump Discharge Pressure
209	High Pressure Oxidizer Pump Inlet Pressure
231	High Pressure Fuel Turbine Discharge Temperature
233†	High Pressure Oxidizer Turbine Discharge Temperature
327	High Pressure Oxidizer Pump Balance Cavity Pressure
480	Oxidizer Preburner Chamber Pressure
734	Low Pressure Oxidizer Pump Shaft Speed
951	High Pressure Oxidizer Pump Primary Seal Drain Pressure
1050	Oxidizer Tank Discharge Temperature
1058	Engine Oxidizer Inlet Temperature
1205	Facility Fuel Flow
1212	Facility Oxidizer Flow
O/Cs	Dummy Parameter indicating Open/Closed Loop Operation
OPBs	Dummy Parameter indicating Oxidizer Preburner Prime Time

† the modeled parameter

same as that of the reference list. Furthermore, in all cases, the genetic algorithm generated parameter lists are smaller than the reference list. Thus, the genetic algorithm was able to systematically generate physically reasonable parameter lists that performed well without the explicit use of problem domain knowledge.

Many improvements for the input selection process have been envisioned. One may, for example, modify the fitness evaluation function to be dependent on the error of a validation set instead of the training set. This would favor parameter lists that yield networks with superior generalizing capabilities instead of lists that yield rapid learning.

As demonstrated by the GA-1 list, smaller size can be overemphasized compared to the error performance. Instead of favoring a parameter list of the smallest size, a list of a particular size could be favored (*e.g.*, the largest tolerable size). This would favor the inclusion of sufficient information while discouraging the use of parameters that

Table 3: Error Statistics from Parameter List GA-1

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	3.787033	0.000322	2.2330
B1060	T	14.743364	0.001223	4.8150
B1061	V	20.168583	0.001657	10.4348
B1062	V	34.029559	0.002832	9.6225
B1063	V	39.671779	0.003301	6.9063
B1066	V	30.608499	0.002532	7.5330
B1067	V	42.103255	0.003498	9.2189
B1070	T	11.699498	0.000945	3.1922
B1071	V	63.607371	0.005154	20.8187
B1072	V	23.816642	0.001898	8.3420
B1075	V	20.268258	0.001669	10.0018
B1077	T	12.931541	0.001045	5.3681

Table 4: Error Statistics from Parameter List GA-2

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	3.341027	0.000284	2.0253
B1060	T	6.059692	0.000503	3.1339
B1061	V	19.080619	0.001568	5.9461
B1062	V	37.601837	0.003129	9.9597
B1063	V	35.212338	0.002930	6.6999
B1066	V	33.799122	0.002796	7.3425
B1067	V	36.724494	0.003051	7.9440
B1070	T	10.692421	0.000864	3.6021
B1071	V	48.479267	0.003929	15.9965
B1072	V	17.781945	0.001417	5.1814
B1075	V	35.017457	0.002884	11.4959
B1077	T	7.973934	0.000644	2.6040

Table 5: Error Statistics from Parameter List GA-3

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	4.015642	0.000341	1.7042
B1060	T	6.114787	0.000507	2.5343
B1061	V	20.477665	0.001682	6.2484
B1062	V	40.542411	0.003374	10.5837
B1063	V	38.320758	0.003188	7.1349
B1066	V	38.782970	0.003208	8.8021
B1067	V	39.245907	0.003261	8.4381
B1070	T	10.516996	0.000850	3.2462
B1071	V	53.008396	0.004296	18.9413
B1072	V	17.990471	0.001434	4.7040
B1075	V	33.172788	0.002732	12.0369
B1077	T	6.889614	0.000557	2.3684

Table 6: Error Statistics from Parameter List REF

Test Firing	*Training/ Validation*	*MSE*	*NMSE*	*Max. % Error*
B1046	T	6.652181	0.000565	3.8462
B1060	T	7.375382	0.000612	3.0370
B1061	V	22.370471	0.001838	4.8509
B1062	V	23.747774	0.001976	7.2832
B1063	V	28.076726	0.002336	7.9618
B1066	V	16.538060	0.001368	7.6115
B1067	V	20.482848	0.001702	6.6011
B1070	T	6.588053	0.000532	3.8668
B1071	V	50.654580	0.004105	11.0878
B1072	V	42.897089	0.003419	6.7544
B1075	V	25.213499	0.002077	9.4449
B1077	T	7.809484	0.000631	4.5456

Table 7: Summary of Parameter List Performance on Training Data

Parm. List	*MSE*		*NMSE*		*Max.*	
	μ	σ	μ	σ	μ	σ
GA-1	10.790359	4.833357	0.000884	0.000392	3.902086	1.445958
GA-2	7.016768	3.101272	0.000574	0.000244	2.841333	0.679829
GA-3	6.884260	2.709112	0.000564	0.000212	2.463281	0.633292
REF	7.106275	0.589258	0.000585	0.000045	3.823920	0.617108

Table 8: Summary of Parameter List Performance on Validation Data

Parm. List	*MSE*		*NMSE*		*Max.*	
	μ	σ	μ	σ	μ	σ
GA-1	34.284241	14.485731	0.002818	0.001180	10.359750	4.397353
GA-2	32.962132	10.068242	0.002713	0.000832	8.820735	3.563816
GA-3	35.192673	11.349328	0.002897	0.000937	9.611175	4.430940
REF	28.747631	11.808645	0.002353	0.000932	7.699442	1.889502

do not significantly improve the error performance.

V. ACKNOWLEDGMENTS

The public domain genetic algorithm GENESIS Version 5.0, written by John J. Grefenstette, was used for the work described in this paper. Furthermore, the fitness evaluation function is a highly modified and optimized derivative of Terry Regier's implementation of the QuickProp training algorithm.

REFERENCES

[1] S. Chen, S. A. Billings, and P. M. Grant. Non-Linear Systems Identification Using Neural Networks. Research Report 370, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, August 1989.

[2] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[3] K. Funahashi. On the Approximate Realization of Continuous Mappings by Neural Networks. *Neural Networks*, 2:183–192, 1989.

[4] Claudia M. Meyer and William A. Maul. The Application of Neural Networks to the SSME Startup Transient. AIAA 91-2530, July 1991.

[5] D. K. Makel, W. H. Flaspohler, and T. W. Bickmore. Sensor Data Validation and Reconstruction, Phase 1: System Architecture Study. NASA CR 187122, 1991. Contract No. NAS3-25883.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[7] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[8] David J. Powell, Michael M. Skolnick, and Siu Shing Tong. Interdigitation: A Hybrid Technique for Engineering Design Optimization Employing Genetic Algorithms, Expert Systems, and Numerical Optimization. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 20, pages 312–331. Van Nostrand Reinhold, New York, 1991.

[9] Scott E. Fahlman. Faster-Learning Variations on Back-Propagation: An Empirical Study. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, San Mateo, CA, June 1988. Carnegie Mellon University, Morgan Kaufmann Publishers.

[10] Charles C. Peck, Atam P. Dhawan, and Claudia M. Meyer. SSME Parameter Modeling using Neural networks and Genetic Algorithm based Input Selection. Technical Report TR_141/1/93/ECE, Dept. of Elect. and Comp. Eng., Univ. of Cincinnati, Dept. of Elect. and Comp. Eng., Univ. of Cincinnati, Cincinnati, OH 45221, January 1993.